STUDIA INFORMATICA 2013
Volume 34 Number 2A (111)

Ewa PŁUCIENNIK-PSOTA
Silesian University of Technology, Institute of Computer Science

OBJECT (NOT ONLY) RELATIONAL INTERFACES

Summary. NoSQL databases boldly step into the area reserved so far for relational databases. NoSQL databases are more flexible than relational ones at cost of some disadvantages related especially with security and integrity constraints. The most convenient way to access a relational database in modern object applications are object-relational mapping tools. It turns out that some of these tools offer mappings to broad spectrum of NoSQL and object databases and text files. The aim of this article is to present this kind of tools and answer the question if they are really universal i.e. if they allow to persist, in any form, the same object entity as defined at the level of an object application.

Keywords: object application, relational database, NoSQL database, object database, object-relational mapping, impedance mismatch, polyglot persistence, JPA, JDO, DataNucleus, Hibernate Object/Grid Mapper

INTERFEJSY OBIEKTOWO (NIE TYLKO) RELACYJNE

Streszczenie. Bazy danych NoSQL coraz śmielej wkraczają na obszary zarezerwowane do tej pory dla relacyjnych baz danych. Bazy takie są bardziej elastyczne od baz relacyjnych, co okupione jest pewnymi wadami, związanymi szczególnie z bezpieczeństwem i więzami integralności. W dzisiejszych aplikacjach obiektowych najwygodniejszym sposobem dostępu do relacyjnej bazy danych jest zastosowanie narzędzi odwzorowania obiektowo-relacyjnego. Okazuje się, że niektóre z tych narzędzi oferują również odwzorowanie do szerokiej gamy baz NoSQL, baz obiektowych oraz plików tekstowych. Celem artykułu jest zaprezentowanie tego typu narzędzi i odpowiedź na pytanie, do jakiego stopnia są one uniwersalne, tzn. czy pozwalają tę samą encję trwałą zdefiniowaną na poziomie aplikacji obiektowej utrwalić w dowolny sposób.

Słowa kluczowe: aplikacja obiektowa, relacyjna baza danych, baza danych NoSQL, obiektowa baza danych, odwzorowanie obiektowo-relacyjne, niezgodność impedancji, trwałość heterogeniczna, JPA, JDO, DataNucleus, Hibernate Object/Grid Mapper

1. Introduction

Every non trivial application cooperates with a database. In most cases the application is object-oriented and the database is relational. But this schema changes. NoSQL databases boldly step into the area reserved so far for relational databases. NoSQL market is expected to dynamically grow in 2013-2018 [1]. NoSQL databases can be divided into following categories: key-value databases, document databases, column-family databases and graph databases [2]. NoSQL databases gain in importance in cloud computing. The main players on this market, Amazon and Google, use NoSQL databases, SimpleDB and BigTable respectively. Even Microsoft has its own NoSQL 'cloud' database Windows Azure Table [3].

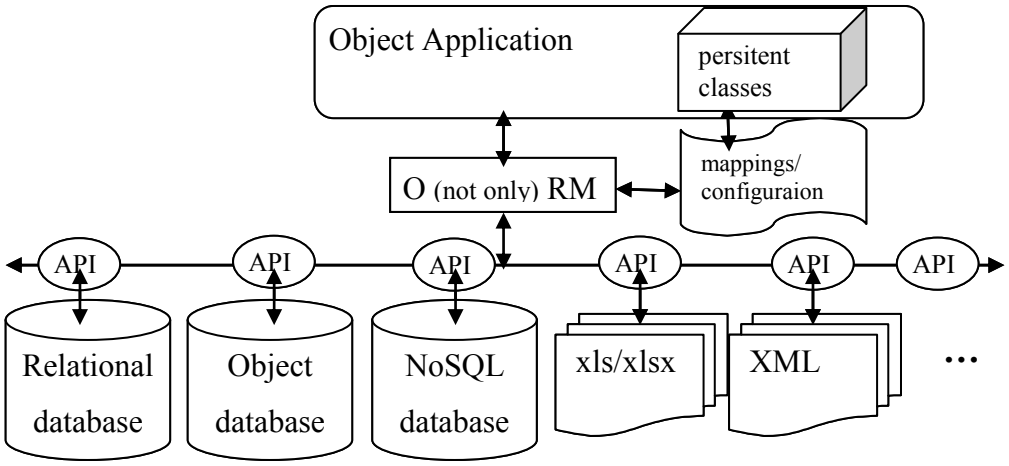

Fig. 1. Scheme of O (not only) RM functioning
Rys. 1. Schemat działania O (nie tylko) RM

The most convenient way to access a relational database in modern object applications is an object-relational mapping tools. It turns out that some of these tools try to meet NoSQL market and user demands and provide unified interface to access data stored in different formats: relational and NoSQL databases and also *xls*, XML files or LDAP. In perfect world universal object-any storage format mapping tool would function like it is shown in Fig. 1. From the point of view of the object application there exists a package of persistent classes and an additional layer which is able to store/retrieve objects of these classes from any desired data storage with uniform methods. In consequence company can store its data in the most appropriate format depends on the data characteristic.

An idea of standardized access for different data stores is called Polyglot Persistence [2]. This is not the new idea. In the year 2000 appeared Java Data Objects Specification as Java Specification Request no. 12 [4]. JDO is a big brother of Java Persistence API which first specification appeared as a part of JSR 220 in 2004 [5]. The main difference between these

two specification is the data storage support: JPA supports only relational database, JDO any data store. JPA is a subset of possibilities offered by JDO [6].

There exist few JDO implementations, but the most advanced, if it comes to data store diversity, is DataNucleus, which is also a reference implementation of JDO 3.1 [7]. Full list of data stores DataNucleus (the newest version) can cooperate with encompasses: MySQL, PostgreSQL, HSQL DB, H2 Database, SQLite, Apache Derby, Microsoft SQLServer, Sybase, Oracle, IBM DB2, IBM Informix, Firebird, SAPDB/MaxDB, Virtuoso, Pointbase, Oracle, McKoi, Excel, OOXML, ODF, XML, HBase, MongoDB, Google AppEngine/DataStore, Neo4j, JSON, Amazon S3, GoogleStorage, LDAP, NeoDatis, Cassandra, OrientDB [8]. DataNucleus implements also JPA. Although using JDO for processing NoSQL data seems more natural, as for now JPA is far more popular. In the rest of the article focus will be put on JPA.

DataNucleus is not the only 'heterogeneous' JPA implementation. Hibernate OGM (Object/Grid Mapper) is a very young project. As for now it offers cooperation with MongoDB, Infinispan and Ehcache but his creators pose to themselves far more ambitious goals [9].

2. DataNucleus – Usage Example

To evaluate Data Nucleus in action three simple entities from the DataNucleus tutorial for RDBMS[1] will be used: Product, Book and Inventory. Here is classes definition with annotation defining necessary mappings:

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Product implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    long id;
    String name = null;
    String description = null;
    double price = 0.0;
   //getters, setters, constructors
}

@Entity
public class Book extends Product{
    String author=null;
    String isbn=null;
    String publisher=null;
     //getters, setters, constructors
}

@Entity
public class Inventory implements Serializable {
    @Id
    String name = null;

    @OneToMany(cascade=CascadeType.ALL)
```

[1] http://www.datanucleus.org/products/datanucleus/guides/jpa/tutorial.html

```
    Set<Product> products = new HashSet();
    //getters, setters, constructors
}
```

Above classes code is characteristic for persistent classes used with Object-Relational Mapping tools which are the JPA implementations. We try to persist these entities in six different data stores: MySQL, db4o (object database), MongoDB (NoSQL document database), Neo4j (graph database), *xlsx* and XML files. All six data sources must be defined in *persistence.xml* file (which is also characteristic for JPA) as separate persistence units:

```
<persistence-unit name="Store_1_db4o">
 <provider>org.datanucleus.api.jpa.PersistenceProviderImpl</provider>
 <class>entities.Product</class>
 <class>entities.Book</class>
 <class>entities.Inventory</class>
 <properties>
  <property name="datanucleus.ConnectionURL" value="db4o:file:db_DN_test.db4o"/>
 </properties>
</persistence-unit>
<persistence-unit name="Store_2_MySQL">
 <class>...</class>
 <properties>
  <property name="datanucleus.ConnectionDriverName"
                                              value="com.mysql.jdbc.Driver"/>
  <property name="datanucleus.ConnectionURL"
              value="jdbc:mysql://localhost/dn_lv?characterEncoding=utf-8"/>
  …
 </properties>
</persistence-unit>
<persistence-unit name="Store_3_XLSX">
 <class>...</class>
 <properties>
  <property name="datanucleus.ConnectionURL" value="ooxml:file:dnFile.xlsx"/>
 </properties>
</persistence-unit>
<persistence-unit name="Store_4_XML">
 <class>entities.ProductXML</class>
 <class>entities.BookXML</class>
 <class>entities.InventoryXML</class>
 <properties>
  <property name="datanucleus.ConnectionURL" value="xml:file:dnFile.xml"/>
 </properties>
</persistence-unit>
<persistence-unit name="Store_5_Mongo">
 <class>...</class>
 <properties>
  <property name="datanucleus.ConnectionURL" value="mongodb:localhost/MongoDB"/>
 </properties>
</persistence-unit>
</persistence-unit>
<persistence-unit name="Store_6_Neo4j">
 <class>...</class>
 <properties>
  <property name="datanucleus.ConnectionURL" value="neo4j:neoDB"/>
 </properties>
</persistence-unit>
```

It have turned out that such data store diversity does not allow to use common entities definition. First problem is automatic object *id* generation which was defined for Product entity by *@GeneratedValue* annotation. In case of **.xml* and **.xlsx* files it is not supported so eventually unique ids were generated in the application and mentioned above annotation for

Product class definition was removed. Second problem considered only XML files. DataNucleus handles XML files using JAXB[2] which requires primary key to be of type *String*. This is not a good idea when it comes to for example relational database. Therefore separate persistent classes for XML storage were defined, where type of *id* attribute in *ProductXML* was *String*. Although the difference is very slight it has consequences.

After a successful configuration it is possible to store some data in all data stores. Here is an example code snippet for persisting data in db4o object database:

```
EntityManagerFactory emf =
                          Persistence.createEntityManagerFactory("Store_1_db4o");
EntityManager em = emf.createEntityManager();
IdGenerator idGen = new IdGenerator();//application id generator
Book book1 = new Book(...);
book1.setId(idGen.getId());
...
Inventory inventory = new Inventory("Science-Fiction");
inventory.addProduct(book1);
inventory.addProduct(book2);
inventory.addProduct(book3);
em.persist(inventory);
em.close();
```

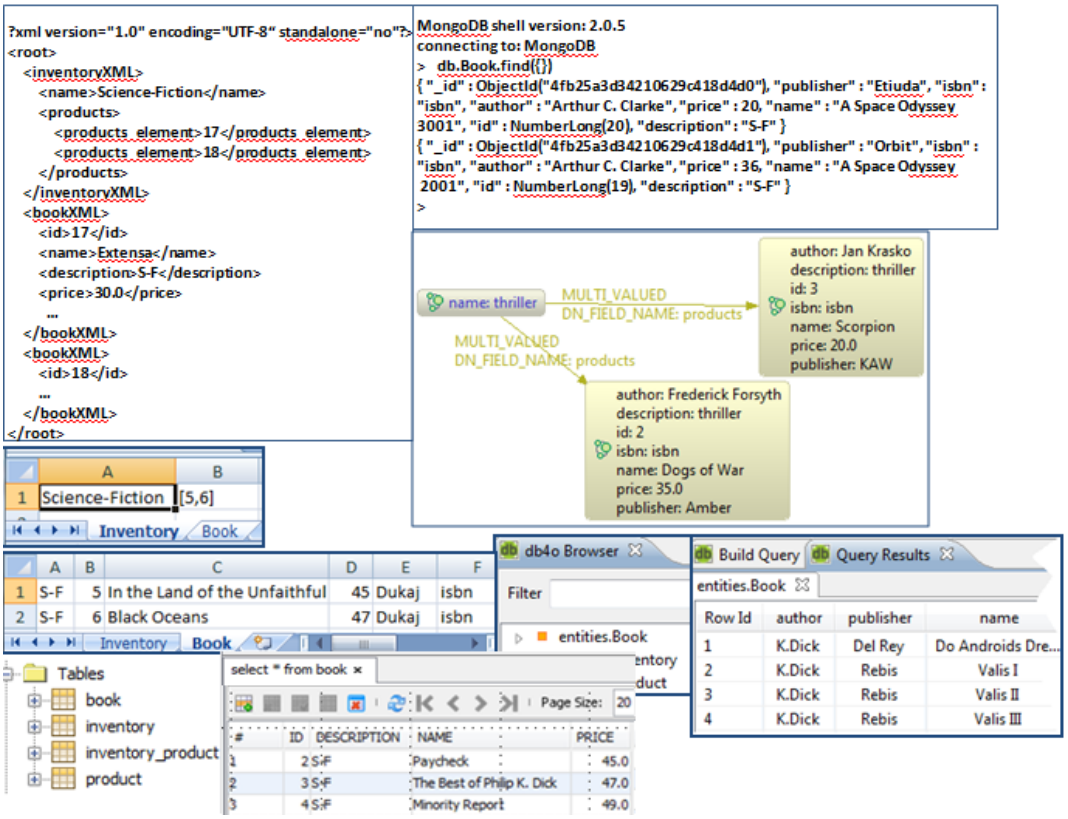

Fig. 2. Entities persisted in different data stores
Rys. 2. Encje utrwalone w różnych strukturach danych

[2] The Java Architecture for XML Binding, http://jaxb.java.net/

The programmer can use chosen data store as in JPA by passing store name defined in *persistence.xml* file to a method that creates entity manager factory. The above code is suitable for all remaining data stores except XML, which uses its own persistent entities definition. Fig. 2 presents entities persisted in different data stores ('native' data stores browsers were used). All persist operations were conducted from the object application using DataNucleus JPA.

So far cooperation with DataNucleus looks promising. Now an attempt to query with JPQL[3] all our data (list of all books) as heterogeneous database can be conducted:

```
//creating EntityManagerFactory for each data store
EntityManagerFactory emf1 =
                         Persistence.createEntityManagerFactory("Store_1_db4o");
EntityManagerFactory emf2 =
                        Persistence.createEntityManagerFactory("Store_2_MySQL");
EntityManagerFactory emf3 =
                         Persistence.createEntityManagerFactory("Store_3_XLXS");
EntityManagerFactory emf4 =
                         Persistence.createEntityManagerFactory("Store_4_XML");
EntityManagerFactory emf5 =
                        Persistence.createEntityManagerFactory("Store_5_Mongo");
EntityManagerFactory emf =
                        Persistence.createEntityManagerFactory("Store_6_Neo4j");
//creating EntityManager for each data store
EntityManager em1 = emf1.createEntityManager();
EntityManager em2 = emf2.createEntityManager();
EntityManager em3 = emf3.createEntityManager();
EntityManager em4 = emf4.createEntityManager();
EntityManager em5 = emf5.createEntityManager();
EntityManager em6 = emf6.createEntityManager();
//query content, different for xml
String query = "select b from entities.Book b";
String queryXML = "select b from entities.BookXML b";
//creating queries for each data store
Query query1 = em1.createQuery(query);
Query query2 = em2.createQuery(query);
Query query3 = em3.createQuery(query);
Query query4 = em4.createQuery(queryXML);
Query query5 = em5.createQuery(query);
Query query6 = em6.createQuery(query);
//getting results from each data store
List<Book> list1 = query1.getResultList();
List<Book> list2 = query2.getResultList();
List<Book> list3 = query3.getResultList();
List<BookXML> list4 = query4.getResultList();
List<Book> list5 = query5.getResultList();
List<Book> list6 = query6.getResultList();
//combining results in one list, for xml results conversion is required
List<Book> list = new ArrayList<Book>(list1);
list.addAll(list2);
list.addAll(list3);
Converter converter = new Converter();
list.addAll(converter.Convert(list4));
list.addAll(list5);
list.addAll(list6);
//printing combined results
for (Book b : list) {
    System.out.println(b.toString());
}
//closing EntityManager for each store
em1.close();
```

[3] Java Persistence Query Language – platform independent query language for JPA

```
em2.close();
em3.close();
em4.close();
em5.close();
em6.close();
```

After looking at the above code, one can see that the programmer must be aware from what source he or she gets the data. The query must be run separately against each data source, but processing is uniform and results are almost uniform.

DataNucleus has different limitations for each data store type. Most of these limitations are well documented in data store supported features matrix [10] and the programmer should be aware of them, although not always. DataNucleus Neo4j plugin is quite fresh and not so well documented, so the programmer has to be prepared for unpleasant surprises. For example, the query:

```
select i.name, count(i.products) from entities.Inventory i inner join i.products
p group by i.name
```

run against Neo4j causes org.neo4j.cypher.EntityNotFoundException: The property 'products' does not exist on Node[1] (Node[1] is *Inventory*). Indeed the *Inventory* node does not have property products. As shown in Fig. 2 1:N relationships between entities persisted in a graph database are represented as edges (type MULTI_VALUED with property DN_FIELD_NAME set to 'products'), not properties. Although DataNucleus creators stipulate that not all JPQL syntax is currently convertible to Cypher (Neo4j query language) [9, 10] the error caused by the query with count function is striking because this query can be converted to Cypher as follows:

```
start i=node(1) match i-[r:MULTI_VALUED]->b
where r.DN_FIELD_NAME = 'products' return i.name, count(b)
```

according to the rule that 1:N relationships are stored as edges of a graph [10]. Nevertheless, the above count query can be successfully run against all five remaining data stores.

DataNucleus is not a perfect solution but is a very interesting proposition for companies which have heterogeneous data sources or use cloud computing (DataNucleus has a plugin for Google App Engine [11]).

3. Hibernate OGM

Assumedly Hibernate OGM will offer 'a familiar programming paradigm to deal with NoSQL, moves model denormalization from a manual imperative work to a declarative approach handled by the engine, encourages new data usage patterns and NoSQL exploration in more "traditional" enterprises and helps scale existing applications with a NoSQL front end

to a traditional database' [9]. A familiar programming paradigm means reusing Hibernate ORM Core. As for now list of NoSQL databases supported by Hibernate OGM encompasses MongoDB, Infinispan and Ehcache. In the following example hibernate OGM is cooperating with MongoDB. The same persistent entities as for DataNucleus tests are used. First the persistence unit definition is needed:

```
<persistence-unit name="Store_Mongo" transaction-type="JTA">
 <provider>org.hibernate.ogm.jpa.HibernateOgmPersistence</provider>
 <class>...</class>
 <properties>
 <property name="hibernate.ogm.datastore.provider"
     value="org.hibernate.ogm.datastore.mongodb.impl.MongoDBDatastoreProvider"/>
 <property name="hibernate.ogm.mongodb.database" value="MongoDB"/>
 <property name="hibernate.ogm.mongodb.host" value="localhost"/>
 <property name="hibernate.ogm.mongodb.port" value="27017"/>

 <property name="hibernate.search.Rules.directory_provider" value="ram"/>
 <property name="hibernate.search.default.directory_provider"
                                                        value="filesystem"/>
 <property name="hibernate.search.default.indexBase"
                                                  value="/var/lucene/indexes"/>
 </properties>
</persistence-unit>
```

What differs this file from standard Hibernate JPA *persistence.xml* file is above all the persistence provider. Properties *hibernate.ogm.mongodb.** allow to configure data store connection. Additional properties apply to the Hibernate Search library configuration.

CRUD[4] operations for entities are the same as for DataNucleus. Regrettably, Hibernate OGM does not yet support JPQL queries, Criteria queries nor native queries. It does not mean that querying data is impossible. We can do it using Hibernate Search library [9]. Hibernate Search provides full-text search capabilities to Hibernate using full-featured text search engine library Apache Lucene [12].

Below code creates and executes full text query which returns all books:

```
EntityManagerFactory emf =
                        Persistence.createEntityManagerFactory("Store_Mongo");
EntityManager em = emf.createEntityManager();
FullTextEntityManager ftem = Search.getFullTextEntityManager(em);
QueryBuilder qb = ftem.getSearchFactory()
                .buildQueryBuilder()
                .forEntity(Book.class)
                .get();
Query lq = qb.all().createQuery();
FullTextQuery ftQuery = ftem.createFullTextQuery(lq, Book.class);
ftQuery.initializeObjectsWith(ObjectLookupMethod.SKIP,
                                          DatabaseRetrievalMethod.FIND_BY_ID);
List<Book> result = ftQuery.getResultList();
em.close();
```

First one need to obtain *FullTextEntityManager* and *QueryBuilder qb*. Using *qb* one can create a Lucene query *lq* and then transform it to a JPA query *ftQuery*. In the end *ftQuery* is run against the data store. It is much more complicated that just using JPQL query with

[4] Create, Read, Update and Delete operation

DataNucleus. What is more the query is run against the Lucene index (according to the above persistence unit definition stored in directory */var/lucene/indexes*), instead of the real data store so the appropriate synchronisation needs to be laid on. Synchronisation is done automatically only if all operations on the data are executed using Hibernate OGM, otherwise a manual indices rebuilding is required.

4. Summary

Two briefly described solutions allow to persist the data in NoSQL databases. DataNucleus is a mature product implementing JPA and JDO, while Hibernate OGM is still at the beginning of the road. The gap between these two tools is massive. OGM documentation has sixty pages while DataNucleus one has more than a thousand. Hopefully OGM creators will continue their work using experience and knowledge acquired with Hibernate ORM. Another solution worth mentioning is Eclipse Link JPA for NoSQL databases [13]. Eclipse offers cooperation with MongoDB, XML files, JMS[5], Oracle AQ[6] and Oracle NoSQL database. The Eclipse solution requires specific NoSQL mappings (e.g. *@NoSql* annotation), but as opposed to OGM supports basic JPQL and Criteria queries and also native queries for MongoDB. Although, this article focuses on JPA it has to be stated that using JDO is far more natural while working with heterogeneous data stores. Hopefully these two specification will evolve in one universal solution.

When it comes to Microsoft, LINQ[7] can be used to access NoSQL Windows Azure Table. Some NoSQL databases also offer querying by LINQ by providing appropriate libraries for .NET [14].

BIBLIOGRAPHY

1. NoSQL Market Forecast 2013-2018; http://www.marketresearchmedia.com/?p=568 [online, access 2012-12-30].
2. Fowler M., Sadalage P. J.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, New York 2012.
3. Rosenberg J., Mateos A.: The Cloud at Your Service. The when, how, and why of enterprise cloud computing. Manning Publication Co., Greenwich 2011.

[5] Java Message Service, Message Oriented Middleware
[6] Oracle Advanced Queuing, Message Oriented Middleware
[7] Language INtegrated Query

4. JSR 12: JavaTM Data Objects (JDO) Specification; http://www.jcp.org/en/jsr/detail?id=12 [online, access 2012-12-30].
5. JSR 220: Enterprise JavaBeansTM 3.0; http://www.jcp.org/en/jsr/detail?id=220 [online, access 2012-12-30].
6. http://db.apache.org/jdo/jdo_v_jpa.html [online, access 2013-01-05].
7. JDO implementations; http://db.apache.org/jdo/impls.html [online, access 2013-01-05].
8. http://www.datanucleus.org/products/accessplatform_3_2/datastores/index.html [online, access 2013-01-05].
9. Hibernate Object/Grid Mapper Reference Guide 4.0.0.Beta1; http://www.datanucleus-.org/ products/datanucleus/datastores/neo4j.html [online, access 2013-01-05].
10. DataNucleus AccessPlatform v.3.2 User Guide; http://www.datanucleus.org/products/accessplatform_3_2/datanucleus-accessplatform-docs.pdf [online, access 2013-01-12].
11. DataNucleus plugin for Google App Engine; http://code.google.com/p/datanucleus-appengine/ [online, access 2013-01-05].
12. Bernard E., Griffin J.: Hibernate Search in Action. Manning Publication Co., Greenwich 2009.
13. http://wiki.eclipse.org/EclipseLink/Examples/JPA/NoSQL [online, access 2013-01-05].
14. http://nosql-database.org/ [online, access 2013-01-05].

Wpłynęło do Redakcji 15 stycznia 2013 r.

Omówienie

Obecnie najpopularniejszym sposobem współpracy obiektowej aplikacji z relacyjną bazą danych jest zastosowanie dodatkowej warstwy pośredniczącej w postaci narzędzi odwzorowania obiektowo-relacyjnego. Okazuje się, że niektóre z tych narzędzi oferują także dostęp do innych typów zbiorów danych, np. w formacie XML bądź XLS, a także do baz danych NoSQL. Bazy danych NoSQL coraz śmielej wkraczają na obszary zarezerwowane do tej pory dla relacyjnych baz danych. Bazy takie są bardziej elastyczne od baz relacyjnych, co jest okupione pewnymi wadami, związanymi szczególnie z bezpieczeństwem i więzami integralności. Bazy NoSQL bardzo zyskują na znaczeniu w dobie coraz bardziej popularnego przetwarzania w chmurze. Główne firmy (Amazon, Google czy Microsoft) na tym rynku oferują właśnie bazy NoSQL do przechowywania danych.

W niniejszym artykule przedstawiono dwa narzędzia oferujące ujednolicony dostęp do heterogenicznych źródeł danych. Jednym z nich jest DataNucleus, będący implementacją

JDO i JPA dla: MySQL, PostgreSQL, HSQL DB, H2 Database, SQLite, Apache Derby, Microsoft SQLServer, Sybase, Oracle, IBM DB2, IBM Informix, Firebird, SAPDB/MaxDB, Virtuoso, Pointbase, Oracle, McKoi, Excel, OOXML, ODF, XML, HBase, MongoDB, Google AppEngine/DataStore, Neo4j, JSON, Amazon S3, GoogleStorage, LDAP, NeoDatis, Cassandra, OrientDB. Jest to narzędzie bardzo rozbudowane i, choć dla poszczególnych typów zbiorów danych narzuca pewne ograniczenia, stanowi bardzo interesującą propozycję dla przedsiębiorstw posiadających heterogeniczne zbiory danych. Znakomicie wpisuje się także w zagadnienie trwałości heterogenicznej (ang. *polyglot persistence*), która zakłada, że dane będą przechowywane w najbardziej odpowiednim (np. pod względem wydajności) dla nich formacie. Drugie przedstawione pokrótce narzędzie to Hibernate Object/Grid Mapper, który to projekt jak na razie znajduje się w fazie początkowej, ale ma ambitne cele, które przy wzięciu pod uwagę dziedzictwa Hibernate'a – mają szanse na realizację.

Address

Ewa PŁUCIENNIK-PSOTA: Silesian University of Technology, Institute of Computer Science, Akademicka 16, 44-100 Gliwice, Poland, Ewa.Pluciennik-Psota@polsl.pl.